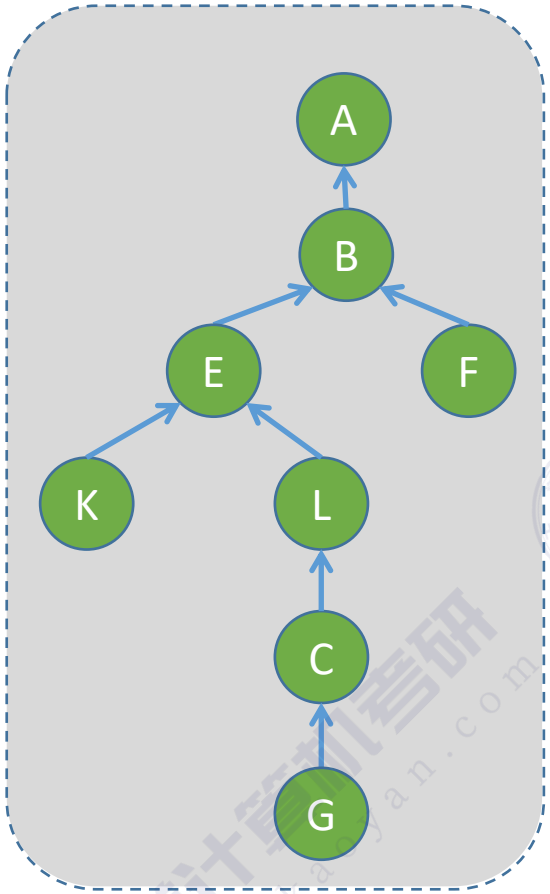
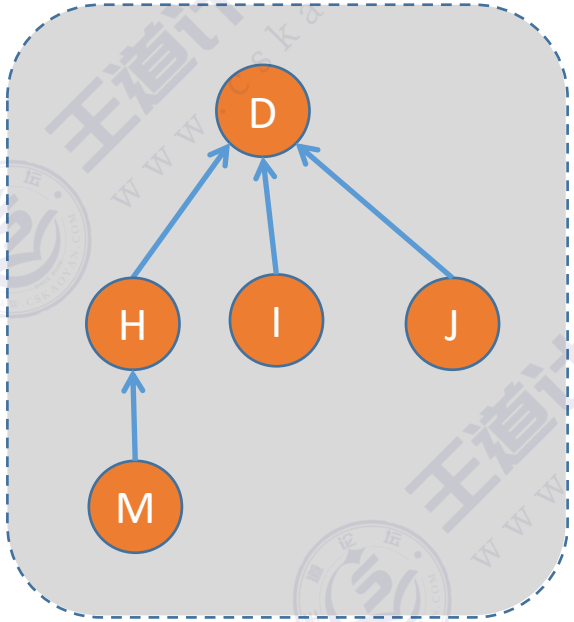
本节内容

补充：并查集的终极优化

年轻人
不讲武德

拓展：Find 操作的优化（压缩路径）





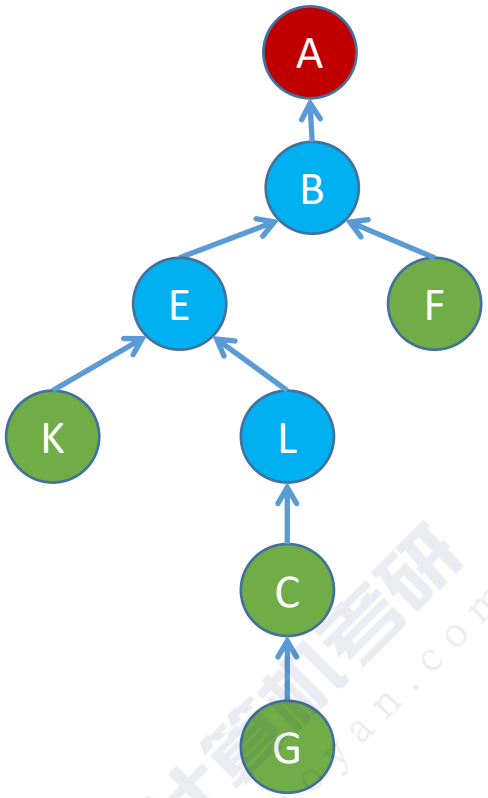

```
//Find “查”操作，找x所属集合（返回x所属根结点）
int Find(int S[],int x){
    while(S[x]>=0)          //循环寻找x的根
        x=S[x];
    return x;               //根的S[]小于0
}
```

压缩路径——Find 操作，先找到根节点，再将查找路径上所有结点都挂到根结点下

数据元素	A	B	C	D	E	F	G	H	I	J	K	L	M
数组下标	0	1	2	3	4	5	6	7	8	9	10	11	12
S[ ]	-8	0	11	-5	1	1	2	3	3	3	4	4	7

Eg：Find(S[ ], 11)　//查找结点L所属集合

拓展：Find 操作的优化（压缩路径）

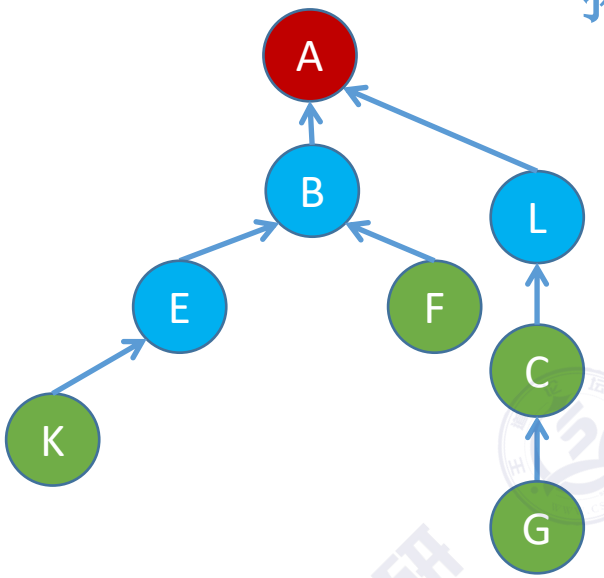


压缩路径——Find 操作，先找到根节点，再将查找路径上所有结点都挂到根结点下

数据元素	A	B	C	D	E	F	G	H	I	J	K	L	M
数组下标	0	1	2	3	4	5	6	7	8	9	10	11	12
S[ ]	-8	0	11	-5	1	1	2	3	3	3	4	4	7

Eg：Find(S[ ], 11)　//查找结点L所属集合

拓展：Find 操作的优化（压缩路径）

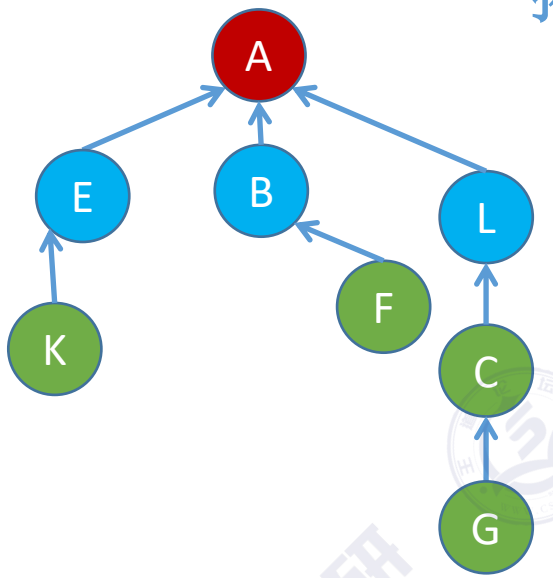


压缩路径——Find 操作，先找到根节点，再将查找路径上所有结点都挂到根结点下

数据元素	A	B	C	D	E	F	G	H	I	J	K	L	M
数组下标	0	1	2	3	4	5	6	7	8	9	10	11	12
S[ ]	-8	0	11	-5	1	1	2	3	3	3	4	0	7

Eg：Find(S[ ], 11)　//查找结点L所属集合

拓展：Find 操作的优化（压缩路径）

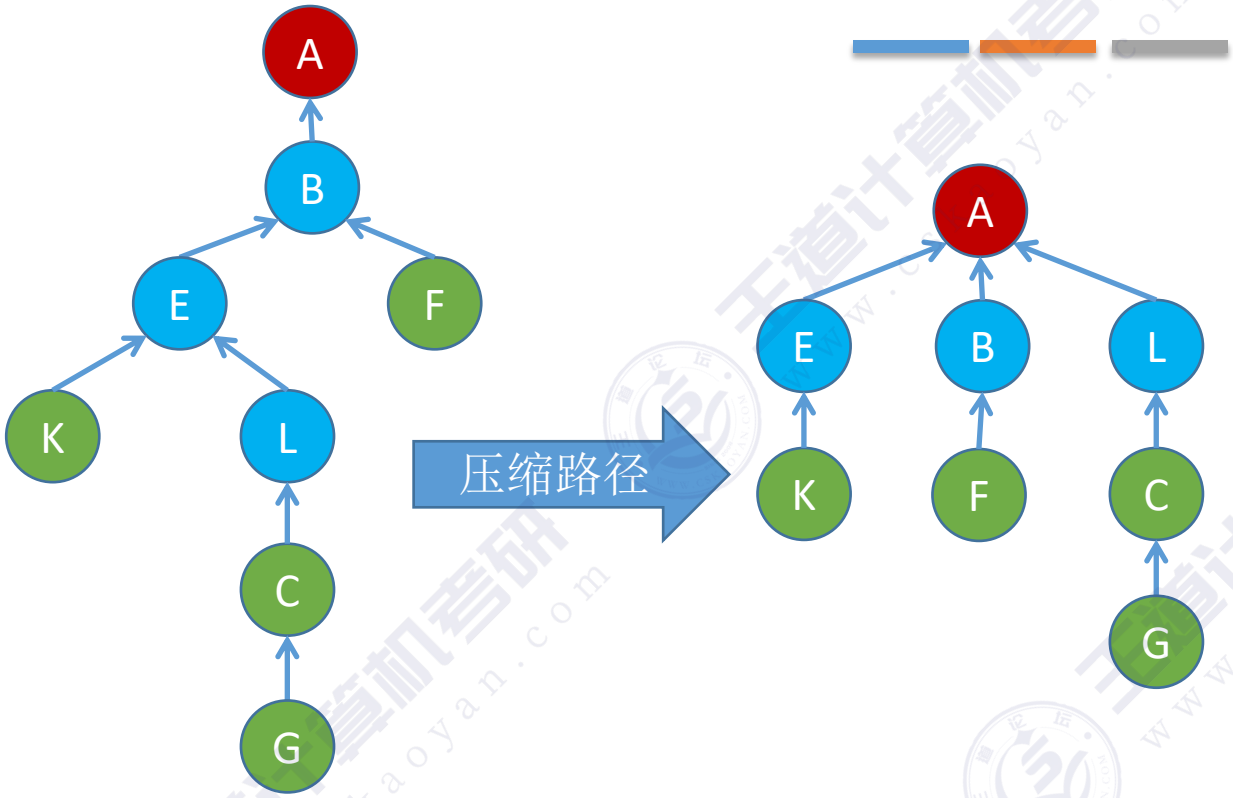


压缩路径——Find 操作，先找到根节点，
再将查找路径上所有结点都挂到根结点下

数据元素	A	B	C	D	E	F	G	H	I	J	K	L	M
数组下标	0	1	2	3	4	5	6	7	8	9	10	11	12
S[ ]	-8	0	11	-5	0	1	2	3	3	3	4	0	7

Eg： Find(S[ ], 11)　　//查找结点L所属集合

拓展：Find 操作的优化（压缩路径）



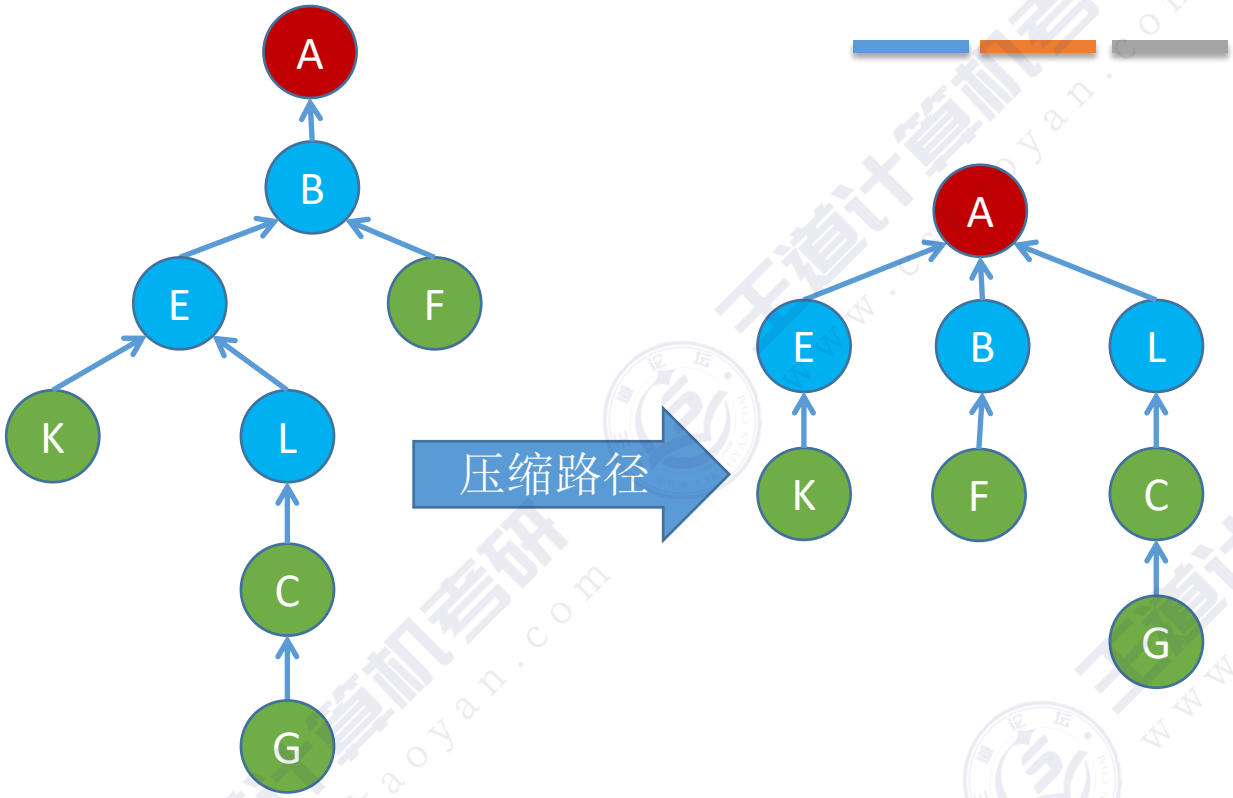

```
//Find "查"操作优化，先找到根节点，再进行"压缩路径"
int Find(int S[],int x){
    int root = x;
    while(S[root]>=0)  root=S[root]; //循环找到根
    while(x!=root){ //压缩路径
      int t=S[x];    //t指向x的父节点
      S[x]=root;     //x直接挂到根节点下
      x=t;
    }
    return root;                    //返回根节点编号
}
```

压缩路径——Find 操作，先找到根节点，再将查找路径上所有结点都挂到根结点下

数据元素	A	B	C	D	E	F	G	H	I	J	K	L	M
数组下标	0	1	2	3	4	5	6	7	8	9	10	11	12
S[ ]	-8	0	11	-5	0	1	2	3	3	3	4	0	7

Eg：Find(S[ ], 11)　//查找结点L所属集合

拓展：Find 操作的优化（压缩路径）




```
//Find “查”操作优化，先找到根节点，再进行“压缩路径”
int Find(int S[],int x){
    int root = x;
    while(S[root]>=0)  root=S[root]; //循环找到根
    while(x!=root){ //压缩路径
      int t=S[x];    //t指向x的父节点
      S[x]=root;     //x直接挂到根节点下
      x=t;
    }
    return root;                  //返回根节点编号
}
```

压缩路径——Find 操作，先找到根节点，再将查找路径上所有结点都挂到根结点下

每次 Find 操作，先找根，再“压缩路径”，可使树的高度不超过$O(\alpha(n))$。$\alpha(n)$是一个增长很缓慢的函数，对于常见的n值，通常$\alpha(n)\leq 4$，因此优化后并查集的Find、Union操作时间开销都很低

并查集的优化

```
#define SIZE 13
int UFSets[SIZE];    //集合元素数组

//初始化并查集
void Initial(int S[]){
    for(int i=0;i<SIZE;i++)
        S[i]=-1;
}
```

```
//Find "查"操作，找x所属集合（返回x所属根结点）
int Find(int S[],int x){
    while(S[x]>=0)        //循环寻找x的根
        x=S[x];
    return x;             //根的S[]小于0
}
```

```
//Union "并"操作，将两个集合合并为一个
void Union(int S[],int Root1,int Root2){
    //要求Root1与Root2是不同的集合
    if(Root1==Root2)  return;
    //将根Root2连接到另一根Root1下面
    S[Root2]=Root1;
}
```

核心思想：尽可能让树变矮

Find优化（压缩路径）

```
//Find "查"操作优化，先找到根节点，再进行"压缩路径"
int Find(int S[],int x){
    int root = x;
    while(S[root]>=0)  root=S[root]; //循环找到根
    while(x!=root){ //压缩路径
      int t=S[x];    //t指向x的父节点
      S[x]=root;     //x直接挂到根节点下
      x=t;
    }
    return root;              //返回根节点编号
}
```

Union优化

```
//Union "并"操作，小树合并到大树
void Union(int S[],int Root1,int Root2){
    if(Root1==Root2)  return;
    if(S[Root2]>S[Root1]) { //Root2结点数更少
        S[Root1] += S[Root2]; //累加结点总数
        S[Root2]=Root1; //小树合并到大树
    } else {
        S[Root2] += S[Root1]; //累加结点总数
        S[Root1]=Root2; //小树合并到大树
    }
}
```

并查集的优化

用数组、双亲表示法来描述元素之间的集合关系。根节点为 -1，非根节点指向父节点下标。

Find (int S[], x) ——找到x所属集合
Union(int S[], int x, int y) —— 将x、y所属集合合并

Union 优化

用根节点的负值表示一棵树的结点总数

每次 Union 操作让小树合并到大树根节点下面

Find 优化

“压缩路径”的策略，每次 Find 操作先找到 x 所属根节点，再将查找路径上的所有结点都直接挂在根节点下面

最坏时间复杂度：

Find 操作 = 最坏树高 = O(n)

将n个独立元素通过多次Union合并为一个集合——$O(n^2)$

最坏时间复杂度：

Find 操作=最坏树高= $O(\log_2 n)$

将n个独立元素通过多次Union合并为一个集合——$O(n\log_2 n)$

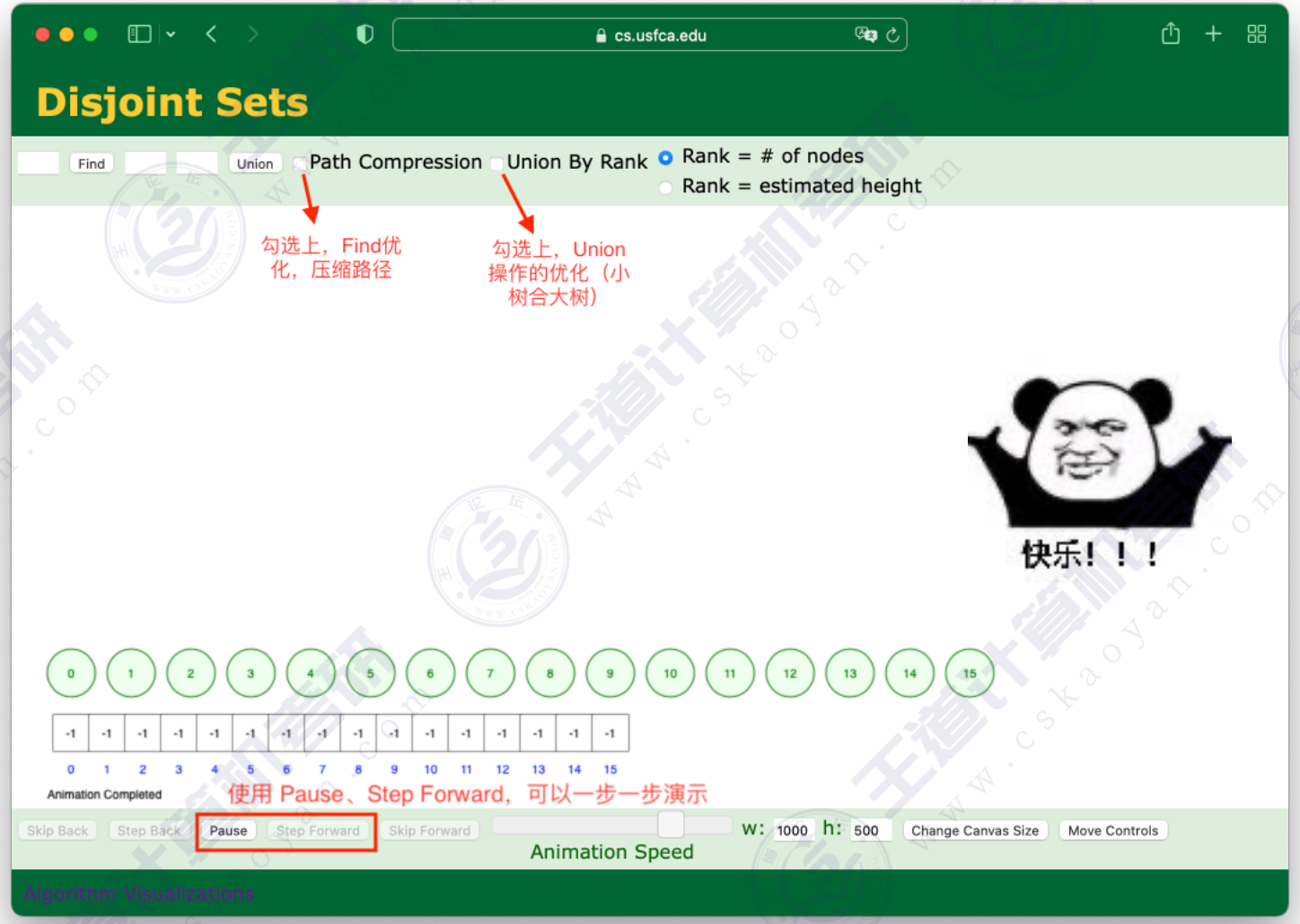
最坏时间复杂度：

Find 操作=最坏树高= $O(\alpha(n))$

将n个独立元素通过多次Union合并为一个集合——$O(n\,\alpha(n))$

408快乐站

点开链接玩一玩：https://www.cs.usfca.edu/~galles/visualization/DisjointSets.html



玩玩各种算法： https://www.cs.usfca.edu/~galles/visualization/Algorithms.html

cs.usfca.edu

David Galles
Computer Science
University of San Francisco

- N-Queens Problem
- Indexing
  - Binary and Linear Search (of sorted list) → 二分查找 和 顺序查找
  - Binary Search Trees → 二叉查找树BST
  - AVL Trees (Balanced binary search trees) → 平衡二叉树AVL
  - Red-Black Trees → 红黑树
  - Splay Trees
  - Open Hash Tables (Closed Addressing) → 散列表（拉链法解决冲突）
  - Closed Hash Tables (Open Addressing) → 散列表（开放定址法解决冲突）
  - Closed Hash Tables, using buckets
  - Trie (Prefix Tree, 26-ary Tree)
  - Radix Tree (Compact Trie)
  - Ternary Search Tree (Trie with BST of children)
  - B Trees
  - B+ Trees → B树、B+树 （注：心中有B树就行，408不要求掌握B+树的插入删除）
- Sorting
  - Comparison Sorting
    - Bubble Sort → 冒泡排序
    - Selection Sort → 选择排序
    - Insertion Sort → 插入排序
    - Shell Sort → 希尔排序
    - Merge Sort → 归并排序
    - Quck Sort → 快速排序
  - Bucket Sort
  - Counting Sort
  - Radix Sort → 基数排序（该网站实现方式和教材中讲授方式不同）
  - Heap Sort → 堆排序
- Heap-like Data Structures
  - Heaps
  - Binomial Queues
  - Fibonacci Heaps
  - Leftist Heaps
  - Skew Heaps
- Graph Algorithms
  - Breadth-First Search → 图的广度优先遍历
  - Depth-First Search → 图的深度优先遍历
  - Connected Components
  - Dijkstra's Shortest Path → 迪杰斯特拉算法
  - Prim's Minimum Cost Spanning Tree → Prim算法求MST
  - Topological Sort (Using Indegree array) → 拓扑排序算法
  - Topological Sort (Using DFS)
  - Floyd-Warshall (all pairs shortest paths) → Floyd算法
  - Kruskal Minimum Cost Spanning Tree Algorithm → Kruskal算法
- Dynamic Programming
  - Calculating nth Fibonacci number
  - Making Change
  - Longest Common Subsequence
- Geometric Algorithms
  - 2D Rotation and Scale Matrices
  - 2D Rotation and Translation Matrices
  - 2D Changing Coordinate Systems
  - 3D Rotation and Scale Matrices
  - 3D Changing Coordinate Systems
- Others ...
  - Disjoint Sets → 并查集
  - Huffman Coding (available in java version)

欢迎大家对本节视频进行评价~

学员评分：5.5.2_2 并...
扫一扫二维码打开或分享给好友

公众号：王道在线

b站：王道计算机教育

抖音：王道计算机考研